Gradient-Based Learning

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics

- Overview

1. Example: Learning XOR
2. Gradient-Based Learning
3. Hidden Units
4. Architecture Design
5. Backpropagation and Other Differentiation
6. Historical Notes

Topics in Gradient-based Learning

- Overview

1. Cost Functions
    1. Learning Conditional Distributions with Max Likelihood
    2. Learning Conditional Statistics
2. Output Units
    1. Linear Units for Gaussian Output Distributions
    2. Sigmoid Units for Bernoulli Output Distributions
    3. Softmax Units for Multinoulli Output Distributions
    4. Other Output Types

Overview of Gradient-based Learning

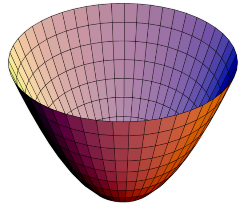
Standard ML Training vs NN Training

- Neural Network training not different from ML models with gradient descent. Need
  1. optimization procedure, e.g., gradient descent
  2. cost function, e.g., MLE
  3. model family, e.g., linear with basis functions
- Difference: nonlinearity causes non-convex loss
  – Use iterative gradient-based optimizers that merely drives cost to low value, rather than
    - Exact linear equation solvers used for linear regression or
    - convex optimization algorithms used for logistic regression or SVMs

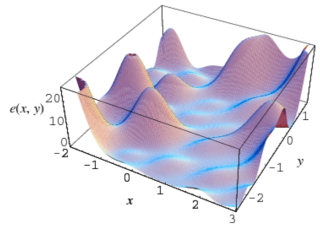
Convex vs Non-convex

Linear Regression with Basis Functions:

$$E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{ t_n - \mathrm{w}^T\varphi(x_n) \right\}^2$$



- Convex methods:
  - Converge from any initial parameters
  - Robust-- but can encounter numerical problems
- SGD with non-convex:



  - Sensitive to initial parameters
  - For feedforward networks, important to initialize
    - Weights to small values, Biases to zero or small positives
  - SGD can also train Linear Regression and SVM Especially with large training sets
  - Training neural net no similar to other models
    - Except computing gradient is more complex

Cost Functions

Cost Functions for Deep Learning

- Important aspect of design of deep neural networks is the cost function
  - They are similar to those for parametric models such as linear models
- Parametric model: logistic regression $p(C_1 \mid \phi) = y(\phi) = \sigma(\theta^T \phi)$
  - Binary Training data defines a likelihood $p(\boldsymbol{y} \mid \boldsymbol{x}\,;\boldsymbol{\theta})$

    $p(\mathbf{t} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} y_n^{t_n} \left\{1 - y_n\right\}^{1-t_n}, y_n = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}_n)$ data set $\{\phi_n, t_n\}, t_n \varepsilon \{0,1\}, \phi_n = \phi(x_n)$
  - and we use the principle of maximum likelihood

    $$J(\boldsymbol{\theta}) = -\ln p(\mathbf{t} \mid \boldsymbol{\theta}) = -\sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}$$
    - Cost function: cross-entropy between training data t_n and the model's prediction y_n
    - Gradient of the error function is $\nabla J(\theta) = \sum_{n=1}^{N} \left(y_n - t_n\right) \phi_n$

      Using $d\sigma(a)/da = \sigma(1-\sigma)$

Learning Conditional Distributions with maximum likelihood

- Specifying the model $p(\boldsymbol{y} \mid \boldsymbol{x})$ automatically determines a cost function $\log p(\boldsymbol{y} \mid \boldsymbol{x})$
  - Equivalently described as the cross-entropy between the training data and the model distribution

$$\boxed{J(\boldsymbol{\theta}) = -E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})}$$

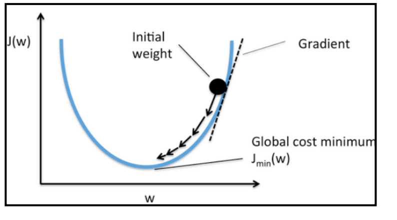
  - Gaussian case:
    - If $p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x}) = N(\boldsymbol{y} \mid f(\boldsymbol{x};\boldsymbol{\theta}), I)$ $\boxed{= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \| \mathrm{y} - f(x;\theta \|^2\right)}$
    - then we recover the mean squared error cost

$$\boxed{J(\boldsymbol{\theta}) = -\frac{1}{2} E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \left\|\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})\right\|^2 + const}$$

    - upto a scaling factor ½ and a term independent of $\boldsymbol{\theta}$
      - $const$ depends on the variance of Gaussian which we chose not to parameterize

Desirable Property of Gradient

- Recurring theme in neural network design is:
  - Gradient must be large and predictable enough to serve as good guide to the learning algorithm
- Functions that saturate (become very flat) undermine this objective
  - Because the gradient becomes very small
    - Happens when activation functions producing output of hidden/output units saturate



Keeping the Gradient Large

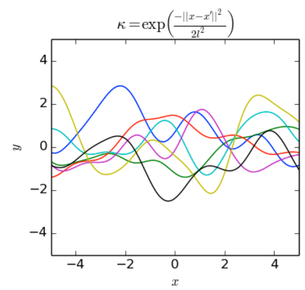
- Negative log-likelihood helps avoid saturation problem for many models
  - Many output units involve exp functions that saturate when its argument is very negative
  - Log function in Negative log likelihood cost function undoes exp of some units

Cross Entropy and Gradient

- A property of cross-entropy cost used for MLE is that it does not have a minimum value
  - For discrete output variables, they cannot represent probability of zero or one but come arbitrarily close
    - Logistic Regression is an example
  - For real-valued output variables it becomes possible to assign extremely high density to correct training set outputs, e.g, variance parameter of Gaussian output, and cross-entropy approaches negative infinity
- Regularization modifies learning problem so model cannot reap unlimited reward this way

Learning Conditional Statistics

- Instead of learning a full probability distribution, learn just one conditional statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$
  - E.g., we may have a predictor $f\,(\boldsymbol{x}\;;\boldsymbol{\theta})$ which gives the mean of $\boldsymbol{y}$
- Think of neural network as being powerful to determine any function f
  - This function is limited only by
    - boundedness and
    - continuity
    - rather than by having a specific parameteric form
  - From this point of view, cost function is a functional rather than a function

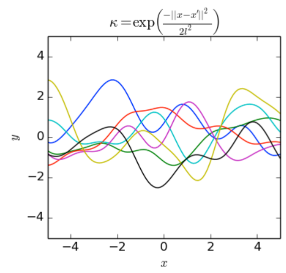


Cost Function vs Cost Functional

- Cost function is a functional, not a function
  - A functional is a mapping from functions to real nos.
- We can think of learning as a task of choosing a function rather than a set of parameters
- Cost Functional has a Minimum occur at a function we desire
  - E.g., Design the cost functional to have a Minimum of that lies on function that maps $\boldsymbol{x}$ to the expected value of $\boldsymbol{y}$ given $\boldsymbol{x}$

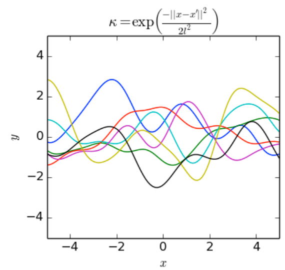
Optimization via Calculus of Variations

- Solving the optimization problem requires a mathematical tool: calculus of variations



  - E.g., Minimum of Cost functional is:
    - function that maps $\boldsymbol{x}$ to the expected value of $\boldsymbol{y}$ given $\boldsymbol{x}$
- Only necessary to understand that calculus of variations can be used to derive two results

First Result from Calculus of Variations



- Solving the optimization problem

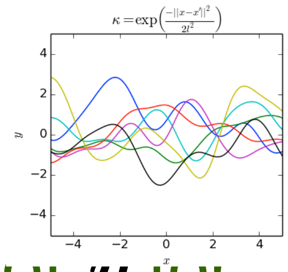
$$\boxed{f^* = \arg\min_f E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \left\| \boldsymbol{y} - \boldsymbol{f(x)} \right\|^2}$$

yields

$$\boxed{f^*(\boldsymbol{x}) = E_{\boldsymbol{y}\sim p_{data}(\boldsymbol{y}|\boldsymbol{x})}\left[\boldsymbol{y}\right]}$$

- which means if we could train infinitely many samples from the true data generating distribution
  - minimizing MSE gives a function that predicts the mean of $\boldsymbol{y}$ for each value of $\boldsymbol{x}$

Second Result from Calculus of Variations



- A different cost function is

$$\boxed{f^* = \arg\min_f E_{x,y\sim p_{data}} \|\boldsymbol{y} - \boldsymbol{f(x)}\|_1}$$

  - yields a function that minimizes the median of $\boldsymbol{y}$ for each each $\boldsymbol{x}$
  - Referred to as *mean absolute error*
- MSE/median saturate: produce small gradients
  - This is one reason cross-entropy cost is more popular than mean square error and mean absolute error
    - Even when it is not necessary to estimate the entire distribution $p(\boldsymbol{y}\,|\boldsymbol{x})$

Output Units

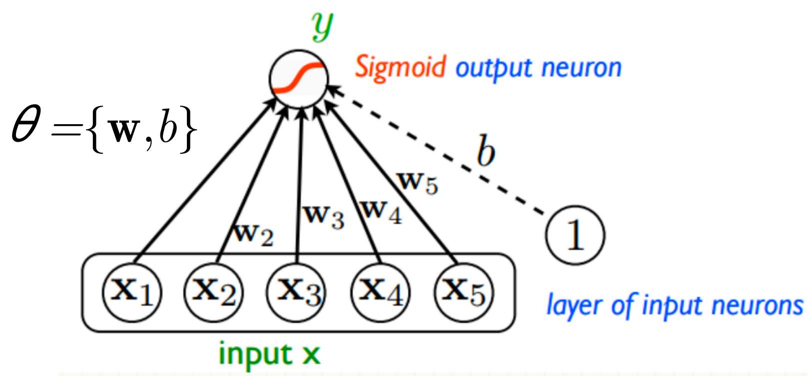
Output Units

- Choice of cost function is *tightly coupled* with choice of output unit
  - Most of the time we use cross-entropy between data distribution and model distribution
    - Choice of how to represent the output then determines the form of the cross-entropy function

Cross-entropy in logistic regression

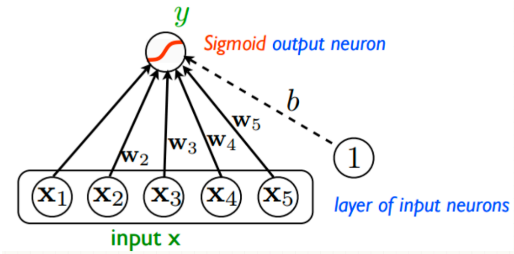
$$
\begin{aligned}
J(\boldsymbol{\theta}) &= -\ln p(\boldsymbol{t} \mid \boldsymbol{\theta}) \\
&= -\sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}
\end{aligned}
$$

$$
y_n = \sigma\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n\right)
$$



Role of Output Units

- Any output unit is also usable as a hidden unit



- Our focus is units as output, not internally
  - Revisit it when discussing hidden units
- A feedforward network provides a hidden set of *features* $\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$
- Role of output layer is to provide some additional transformation from the features to the task that network must perform

Types of output units

1. Linear units: no non-linearity
   – for Gaussian Output distributions
2. Sigmoid units
   – for Bernoulli Output Distributions
3. Softmax units
   – for Multinoulli Output Distributions
4. Other Output Types
   – Not direct prediction of y but provide parameters of distribution over y

Linear Units for Gaussian Output Distributions

- Linear unit: simple output based on affine transformation with *no nonlinearity*
  - Given features $\boldsymbol{h}$, a layer of linear output units produces a vector

$$\hat{\boldsymbol{y}} = W^T \boldsymbol{h} + \boldsymbol{b}$$

- Linear units are often used to produce mean $\hat{\boldsymbol{y}}$ of a conditional Gaussian distribution

$$P(\boldsymbol{y} \mid \boldsymbol{x}) = N(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

- Maximizing the output is equivalent to MSE
- Can be used to learn the covariance of a Gaussian too

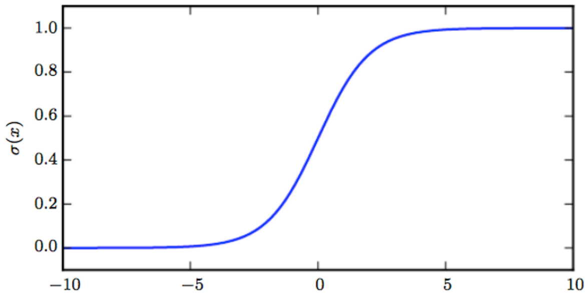
Sigmoid Units for Bernoulli Output Distributions

- Task of predicting value of binary variable y
  - Classification problem with two classes
- Maximum likelihood approach is to define a Bernoulli distribution over y conditioned on $\boldsymbol{x}$
- Neural net needs to predict $p(y=1|\boldsymbol{x})$
  - which lies in the interval $[0,1]$
- Constraint needs careful design
  - If we use $\boxed{P(y=1\,|\,\boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^T\boldsymbol{h}+\boldsymbol{b}\right\}\right\}}$
    - We would define a valid conditional distribution, but cannot train it effectively with gradient descent
    - A gradient of 0: learning algorithm cannot be guided

Sigmoid and Logistic Regression

- Using sigmoid always gives a strong gradient
  - Sigmoid output units combined with maximum likelihood

$$\hat{y} = \sigma\left(\boldsymbol{w}^T \boldsymbol{h} + b\right)$$



  - where $\boldsymbol{\sigma}\ (x)$ is the logistic sigmoid fu

$$\sigma\left(x\right) = \frac{1}{1 + \exp(-x)}$$

- Sigmoid output unit has two components:
  1. A linear layer to compute $z = \boldsymbol{w}^T \boldsymbol{h} + b$
  2. Use sigmoid activation function to convert z into a probability

Probability distribution using Sigmoid

- Describe probability distribution over y using z

$$z = \boldsymbol{w}^T \boldsymbol{h} + b$$

y is output, z is input

  - Construct unnormalized probability distribution $\tilde{P}$
    - Assuming unnormalized log probability is linear in y and z

$$\boxed{\begin{aligned} \log \tilde{P}(y) &= yz \\ \tilde{P}(y) &= \exp(yz) \end{aligned}}$$

    - Normalizing yields a Bernoulli distribution controlled by σ

$$\boxed{\begin{aligned} P(y) &= \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)} \\ &= \sigma((2y-1)z) \end{aligned}}$$

  - Probability distributions based on exponentiation and normalization are common throughout statistical modeling
    - z variable defining such a distribution over binary variables is called a *logit*

Max Likelihood Loss Function

- Given binary y and some z, an normalized probability distribution over y is

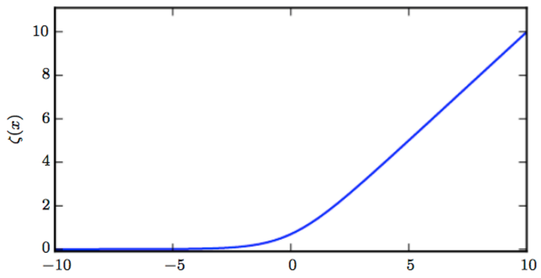
$$\log \tilde{P}(y) = yz$$
$$\tilde{P}(y) = \exp(yz)$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(yz)} = \sigma((2y-1)z)$$

- We can use this approach in maximum likelihood learning
  - Loss for max likelihood learning is $-\log P(y|\boldsymbol{x})$

$$\begin{aligned} J(\theta) &= -\log P(y \mid \boldsymbol{x}) \\ &= -\log \sigma((2y-1)z) \\ &= \zeta((1-2y)z) \end{aligned}$$

ζ is the *softplus* function

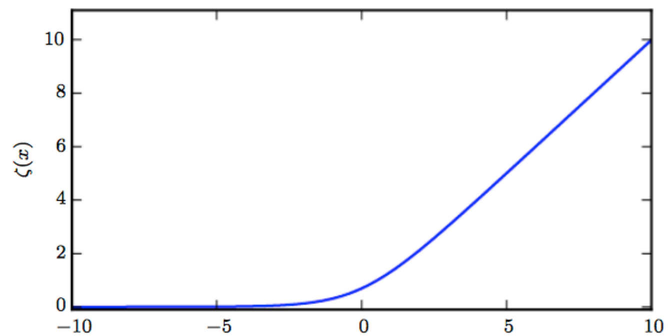


- This is for a single sample

Softplus function

- Sigmoid saturates when its argument is very positive or very negative
  - i.e., function is insensitive to small changes in input
- Another function is the softplus function

$$\zeta(x) = \log(1 + \exp(x))$$



  - Its range is $(0,\infty)$. It arises in expressions involving sigmoids.
- Its name comes from its being a smoothed or softened version of $x^+ = \max(0, x)$

Properties of Sigmoid & Softplus

$$\sigma(x) = \frac{\exp(x)}{\exp(x) + \exp(0)}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

$$1 - \sigma(x) = \sigma(-x)$$

$$\log \sigma(x) = -\zeta(-x)$$

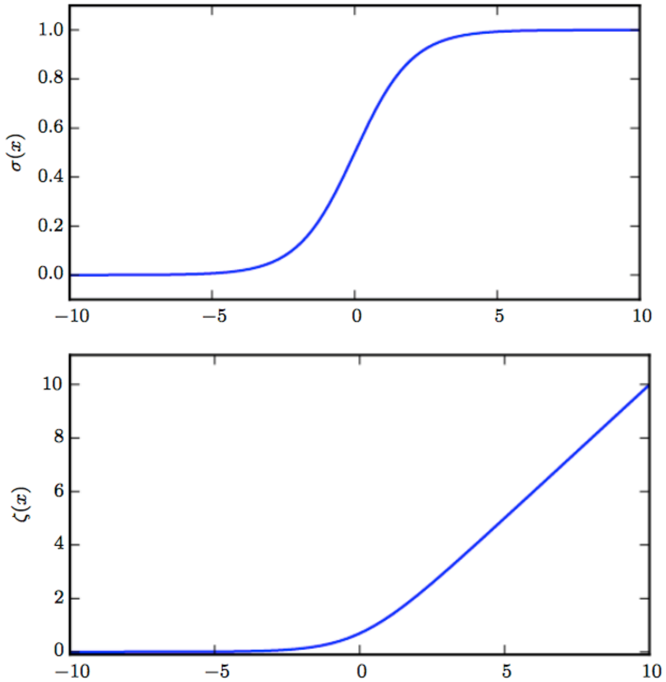
$$\frac{d}{dx}\zeta(x) = \sigma(x)$$

$$\forall x \in (0,1),\ \sigma^{-1}(x) = \log\left(\frac{x}{1-x}\right)$$

$$\forall x > 0,\ \zeta^{-1}(x) = \log\left(\exp(x) - 1\right)$$

$$\zeta(x) = \int_{-\infty}^{x} \sigma(y)dy$$

$$\zeta(x) - \zeta(-x) = x$$



Last equation provides extra justification for the name 'softplus'

Smoothed version of *positive part* function $x^{+}=\max\{0,x\}$

The positive part function is the counterpart of the *negative part* function $x^{-}=\max\{0,-x\}$

Loss Function for Bernoulli MLE

$$\begin{aligned} J(\theta) &= -\log P(y \mid x) \\ &= -\log \sigma((2y-1)z) \\ &= \zeta((1-2y)z) \end{aligned}$$

– By rewriting the loss in terms of the softplus function, we can see that it saturates only when $(1\text{-}2y)z$ <<0.

– Saturation occurs only when model already has the right answer

- i.e., when y=1 and z>>0 or y=0 and z <<0
- When z has the wrong sign $(1\text{-}2y)z$ can be simplified to $|z|$
  - As $|z|$ becomes large while z has the wrong sign, softplus asymptotes towards simply returning argument $|z|$ & derivative wrt z asymptotes to $\text{sign}(z)$, so, in the limit of extremely incorrect z softplus does not shrink the gradient at all
  - This is a useful property because gradient-based learning can act quickly to correct a mistaken z

Cross-Entropy vs Softplus Loss

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} \sigma(\boldsymbol{\theta}^T \boldsymbol{x}_n)^{y_n} \left\{1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}_n)\right\}^{1-y_n}$$
$$J(\boldsymbol{\theta}) = -\ln p(\boldsymbol{y} \mid \boldsymbol{\theta})$$
$$= -\sum_{n=1}^{N} \left\{ y_n \ln\left(\sigma(\boldsymbol{\theta}^T \boldsymbol{x}_n)\right) + (1 - y_n) \ln(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}_n)) \right\}$$

$$J(\theta) = -\log P(y \mid x)$$
$$= -\log \sigma((2y-1)z)$$
$$= \zeta((1 - 2y)z)$$

$$z = \boldsymbol{\theta}^T \boldsymbol{x} + b$$

– Cross-entropy loss can saturate anytime $\sigma(z)$ saturates
  - Sigmoid saturates to 0 when z becomes very negative and saturates to 1 when z becomes very positive

– Gradient can shrink to too small to be useful for learning, whether model has correct or incorrect answer

– We have provided an alternative implementation of logistic regression!

Softmax units for Multinoulli Output

- Any time we want a probability distribution over a discrete variable with n values we may us the *softmax* function
  - Can be seen as a generalization of sigmoid function used to represent probability distribution over a binary variable
- Softmax most often used for output of classifier to represent distribution over n classes
  - Also inside the model itself when we wish to choose between one of n options

From Sigmoid to Softmax

- Binary case: we wished to produce a single no.

$$\boxed{\hat{y} = P(y=1 \mid \boldsymbol{x})}$$

  - Since (i) this number needed to lie between 0 and 1 and (ii) because we wanted its logarithm to be well-behaved for a gradient-based optimization of log-likelihood, we chose instead to predict a number

$$\boxed{z = \log \tilde{P}(y=1 \mid \boldsymbol{x})} \qquad z = \boldsymbol{w}^T \boldsymbol{h} + b$$

  - Exponentiating and normalizing, gave us a Bernoulli distribution controlled by the sigmoidal transformation of z

$$\boxed{\begin{aligned}\log \tilde{P}(y) &= yz \\ \tilde{P}(y) &= \exp(yz)\end{aligned}} \qquad \boxed{P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(yz)} = \sigma((2y-1)z)}$$

- Case of n values: need to produce vector $\hat{\boldsymbol{y}}$
  - with values $\boxed{\hat{y}_i = P(y = i \mid \boldsymbol{x})}$

Softmax definition

- We need to produce a vector $\hat{\boldsymbol{y}}$ with values

$$\hat{y}_i = P(y = i \mid \boldsymbol{x})$$

  - We need elements of $\hat{\boldsymbol{y}}$ lie in $[0,1]$ and they sum to 1
- Same approach as with Bernoulli works for Multinoulli distribution
  - First a linear layer predicts unnormalized log probabilities

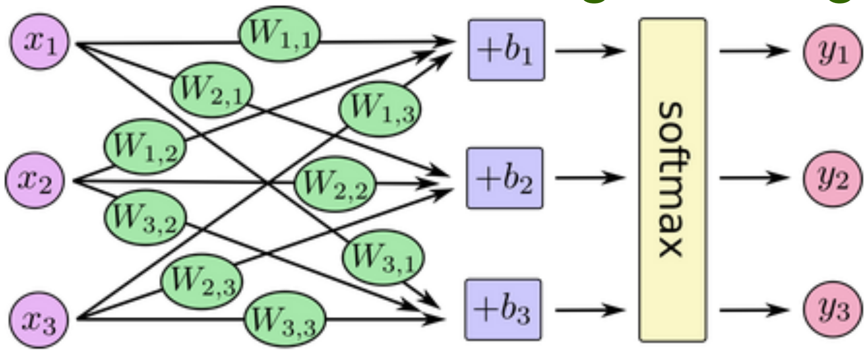
$$\boldsymbol{z} = W^T \boldsymbol{h} + \boldsymbol{b}$$

  – where $\boxed{z_i = \log \hat{P}(y = i \mid \boldsymbol{x})}$
- Softmax can then exponentiate and normalize $\boldsymbol{z}$ to obtain the desired $\hat{\boldsymbol{y}}$
- Softmax is given by:

$$\boxed{\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}}$$

Softmax Regression

Generalization of Logistic Regression to multivalued output



Softmax definition

$$\boldsymbol{y} = \text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Network Computes

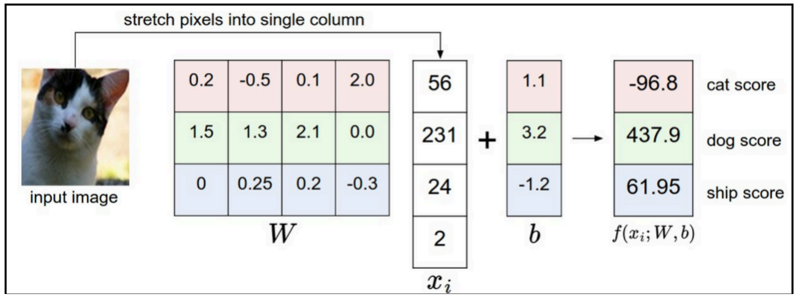
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

In matrix multiplication notation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$

$$\boldsymbol{z} = W^T\boldsymbol{x} + \boldsymbol{b}$$

An example



Intuition of Log-likelihood Terms

- The exp within softmax $\boxed{\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}}$ works very well when training using log-likelihood
  - Log-likelihood can undo the exp of softmax

$$\boxed{\log \text{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)}$$

  - Input z_i always has a direct contribution to cost
    - Because this term cannot saturate, learning can proceed even if second term becomes very small
  - First term encourages z_i to be pushed up
  - Second term encourages all $\boldsymbol{z}$ to be pushed down

Intuition of second term of likelihood

- Log likelihood is $\boxed{\log \operatorname{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)}$
- Consider second term: $\boxed{\log \sum_j \exp(z_j)}$
- It can be approximated by $\max_j z_j$
  - Based on the idea that $\exp(z_k)$ is insignificant for any z_k noticeably less that $\max_j z_j$
- Intuition gained:
  - Cost penalizes most active incorrect prediction
  - If the correct answer already has the largest input to softmax, then $-z_i$ term and $\boxed{\log \sum_j \exp(z_j) \approx \max_j z_j = z_i}$ terms will roughly cancel. This example will then contribute little to overall training cost
    - Which will be dominated by other incorrect examples

Generalization to Training Set

- So far we discussed only a single example
- Overall, unregularized maximum likelihood will drive the model to learn parameters that drive the softmax to predict a *fraction of counts* of each outcome observed in training set

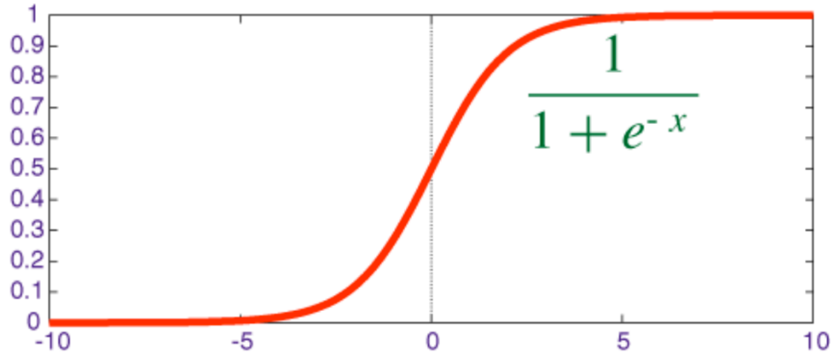
$$\boxed{\mathrm{softmax}(\boldsymbol{z}(\boldsymbol{x};\boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^{m} \mathbf{1}_{y^{(j)}=i,\boldsymbol{x}^{(j)}=\boldsymbol{x}}}{\sum_{j=1}^{m} \mathbf{1}_{\boldsymbol{x}^{(j)}=\boldsymbol{x}}}}$$

Softmax and Objective Functions

- Objective functions that do not use a log to undo the $\exp$ of softmax fail to learn when argument of $\exp$ becomes very negative, causing gradient to vanish
- Squared error is a poor loss function for softmax units
  - Fail to train model change its output even when the model makes highly incorrect predictions

Saturation of Sigmoid and Softmax

- Sigmoid has a single output that saturates
  - When input is extremely negative or positive



- Like sigmoid, softmax activation can saturate
  - In case of softmax there are multiple output values
    - These output values can saturate when the differences between input values become extreme
  - Many cost functions based on softmax also saturate

Softmax & Input Difference

- Softmax invariant to adding the same scalar to all inputs:

$$\mathrm{softmax}(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z}+\mathrm{c})$$

- Using this property we can derive a numerically stable variant of softmax

$$\mathrm{softmax}(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z} - \max_i z_i)$$

- Reformulation allows us to evaluate softmax
  - With only small numerical errors even when $\boldsymbol{z}$ contains extremely large/small numbers
  - It is driven by amount that its inputs deviate from $\max_i z_i$

Saturation of Softmax

- An output $\text{softmax}(\boldsymbol{z})_i$ saturates to 1 when the corresponding input is maximal ($z_i = \max_i z_i$) and z_i is much greater than all the other inputs
- The output can also saturate to 0 when is not maximal and the maximum is much greater
- This is a generalization of the way the sigmoid units saturate
  - They can cause similar difficulties in learning if the loss function is not designed to compensate for it

Other Output Types

- Linear, Sigmoid and Softmax output units are the most common
- Neural networks can generalize to any kind of output layer
- Principle of maximum likelihood provides a guide for how to design a good cost function for any output layer
  - If we define conditional distribution $p(\boldsymbol{y}\,|\boldsymbol{x})$, principle of maximum likelihood suggests we use $\log p(\boldsymbol{y}\,|\boldsymbol{x})$ for our cost function

Determining Distribution Parameters

- We can think of the neural network as representing a function $f\,(\boldsymbol{x}\ ;\boldsymbol{\theta})$
- Outputs are not direct predictions of value of $\boldsymbol{y}$
- Instead $f\,(\boldsymbol{x}\ ;\boldsymbol{\theta})=\omega$ provides the *parameters for a distribution* over $\boldsymbol{y}$
- Our loss function can then be interpreted as $-\log p(\boldsymbol{y}\ ;\ \omega(\boldsymbol{x}))$

Ex: Learning a Distribution Parameter

- We wish to learn the variance of a conditional Gaussian of $\boldsymbol{y}$ given $\boldsymbol{x}$
- Simple case: variance σ^2 is constant
  - Has closed-form expression: empirical mean of squared difference between observations $\boldsymbol{y}$ and their expected value
  - Computationally more expensive approach
    - Does not require writing special-case code
    - Include variance as one of the properties of distribution $p(\boldsymbol{y}\,|\boldsymbol{x})$ that is controlled by $\omega = f\,(\boldsymbol{x}\,;\,\boldsymbol{\theta})$
    - Negative log-likelihood $-\log p(\boldsymbol{y}\,;\,\omega(\boldsymbol{x}))$ will then provide cost function with appropriate terms to learn variance